Interdisciplinary Insights on Open Source

Cristina Gacek, Tony Lawrie, and Budi Arief
Centre for Software Reliability
Department of Computing Science
University of Newcastle
Newcastle upon Tyne NE1 7RU
United Kingdom
{Cristina.Gacek, A.T.Lawrie, L.B.Arief}@ncl.ac.uk

Abstract

The term "open source" is widely applied to describe some software development methodologies. This paper does not provide a judgment on the open source approach, but exposes the fact that simply stating that a project is open source does not provide a precise description of the approach used to support the project. By taking a multi-disciplinary point of view, we propose a collection of characteristics that are common, as well as some that vary among open source projects. The set of open source characteristics we found can be used as a tick-list both for analysing and for setting up open source projects. Our tick-list also provides a starting point for understanding the many meanings of the term open source.

1 Introduction

We started looking into *Open Source* to try to determine how using this approach actually impacts the dependability of the software systems being developed. Our intention was to spend some minor effort to understand what is meant by the term "open source", and from there perform various studies and experiments to support or to oppose dependability claims in the area. Much to our surprise, understanding what open source is turned out to be a much more complex task. The term "open source" has been widely used to describe a software development process that relies on the contribution of its geographically dispersed developers by the means of the Internet. Amongst other criteria, one basic requirement of open source projects is the availability of its source code [1], without which the development or evolution of the software is very difficult if not impossible. But apart from these characteristics, there seems to be some confusion on what actually makes a project an open source project.

The aim of this paper is therefore to provide a clearer description on what is meant by "open source". To achieve this aim, we investigated several well-known open source projects such as Linux [2], Apache [3] and Mozilla [4]. We also did literature studies on published materials about open source, notably *The Cathedral and the Bazaar* [5], *Rebel Code* [6], *Open Sources* [7] as well as work by other people interested on open source (for example, [8-12]). We have also used several on-line resources dedicated to various open source projects [13, 14] and interviewed both individuals working on open source projects at their free time and individuals involved with open source as part of their job in large corporations. From there, we tried to dissect open source further by determining the characteristics that open source projects should or usually have. We determined a set of characteristics that are almost always present and others that vary among open source projects, and this serves as the core of this work.

The rest of this paper is structured as follows: Section 2 presents a brief history of open source, which is important for understanding its motives and directions; Section 3 describes some open source characteristics that can be used in determining whether a project is or not open source; Section 4 provides some initial conclusions of our work; and Section 5 outlines areas that can be researched further.

2 A Brief History of Open Source

2.1 How it started

The idea of building software within a cooperating community, where the source code was made available so that everyone could modify and redistribute it began with the GNU project at MIT in the early 1980s. The intention was to provide *freedom* relating to software systems. In 1985 the *Free Software Foundation (FSF)* was pioneered by Richard Stallman to generate some income for the free software movement, not restricting itself to GNU.

Free software, as defined by the FSF, is a program that grants various freedoms to its users. A free software program provides its users with [15]:

- Freedom to run the program for any purpose
- Freedom to study and adapt the code for personal use
- Freedom to redistribute copies of the program, either gratis or for a fee
- Freedom to distribute improved or modified versions of the program to the public

The discourse used by the FSF tends to be confrontational and against proprietary (closed) software, since they view anyone producing this kind of software as big obstacles to the four basic freedoms mentioned above. This is reflected in the restrictive viral nature of some of their licenses (see section 3.3).

2.2 Free Software and Open Source Movements

In the early 1998, the term *Open Source* was coined as a response to the announcement made by Netscape on its plan to give away the source code of its web browser. The new term came out of a strategy meeting in which people present realised that:

> *"…it was time to dump the confrontational attitude that has been associated with 'free software' in the past and sell the idea strictly on the same pragmatic, business-case grounds that motivated Netscape." [16]*

Immediately afterwards, the *Open Source Initiative (OSI)* was set up to manage and promote the *Open Source Definition (OSD)*. The OSD was composed as a guideline to determine whether a particular software distribution can be called open source or not. OSD asserts nine criteria that open source software must follow; the main three are:

- The ability to distribute the software freely
- The availability of the source code, and
- The right to create derived works through modification.

The rest of the criteria deals with the licensing issues and spell out the "no discrimination" stance that must be followed [1]. They are:

- The integrity of the author's source code must be preserved, making the source of changes clear to the community
- No discrimination against persons or groups both for providing contributions and for using the software
- No restriction on the purpose of usage of the software, providing no discrimination against fields of endeavour
- The rights attached to the software apply to all recipients of its (re)distribution
- The license must not be specific to a product, but apply to all sub-parts within the licensed product
- The license must not contaminate other software, permitting the distribution of other non-open source software along with open source one

The Open Source and Free Software movements can be compared to two political parties within a community. While two political parties agree on the basic principles but disagree on practical issues, the Open Source and Free Software do exactly the opposite. They disagree on the basic principles (commercialism, licensing, etc.), but agree on (most of) practical recommendations (availability of source code, ability to modify the code, etc.). They even work together on many specific projects to achieve the same goal: to provide software that is free (in terms of liberty) for all [17].

2.3 Commercialisation of Open Source

Open source is often seen as a marketing ploy to make Free Software more attractive to business users since it allows greater liberties with its licenses (see section 3.3). This means that the open source licenses are more accommodating to people or companies to make profit from the software, as long as the source code remains available and can be modified freely.

The most prominent way of commercialising open source is by providing service and distribution packages for software developed in an open source fashion. This is due to the fact that open source software is usually more difficult to install since it was originally aimed for the hacker community. Another way of making money out of open source is by using the relevant open source as a platform, upon which commercial (often proprietary) application software can be built.

More and more computing corporations turn their attention to open source as a business opportunity. What they are looking for in this new development method is *innovation*, and sharing source code is perceived to be a good way for facilitating creativity. Commercial organizations are also attracted to contributing to open source projects as they see a strategic opportunity to undermine (more powerful/dominating) competitors. On the down side, they are afraid that maintaining control of an active open source project can be difficult. They are particularly concerned with the risk of code forking – the evolution of two (or more) separate strands of work from the original code base, which threatens compatibility. This fear prevents some individuals and many companies from active participation in open source developments [7].

Although this code forking risk is always present, it is usually overcome by the novel attitude that the open source community has. Instead of basing their reputation on "what they have", they measure it against "what they give". This "gift-culture" encourages people to contribute more and binds people together in the same strand of work. More information on the "gift-culture" is available from Eric Raymond's paper, *Homesteading the Noosphere* [18].

2.4 The Open Source Approach compared with Others

To provide a clearer picture on where open source (free) software stands in relation to other software, we provide some comparisons (mostly in licensing and distribution terms) among several categories of software. For simplicity, we could say that the two main categories are the "free" software (meaning open source as well) and the "proprietary" software.

There are two kinds of software within the "free" category: *non-copylefted free software* and *copylefted software*. Non-copylefted free software comes from the author with permission to modify and redistribute, and in a legal term it means "not copyrighted". On top of that, it is allowed to add more restrictions to the modified version, which means that some copies (modified versions) may not be free at all. Anyone can compile the program and redistribute the binary as proprietary software. *Public domain software* is a special case of non-copylefted free software. On the other hand, with copylefted software, it is not allowed to have additional restrictions to be added when someone redistributes or modifies the software. As a consequence, every copy of copylefted software, even after modification, must be a free software. The most prominent distribution terms for copylefted software are covered in the *GNU GPL (General Public License).*

Proprietary software is *closed* software in that the source code is not available to the public. It has very restrictive terms on its condition of use, and its redistribution or modification is prohibited. There are two special cases within this group of software: *shareware* and *freeware*. Both allow people to download, use and redistribute the software for free, but modification is (almost) impossible because they are usually released in executable (binary) format only. The difference is on the limit of usage, if someone wants to keep using a shareware, he/she must pay a license fee. One important note is that freeware must not be confused with free software, especially because modification of a freeware is not possible (since the source code is not available).

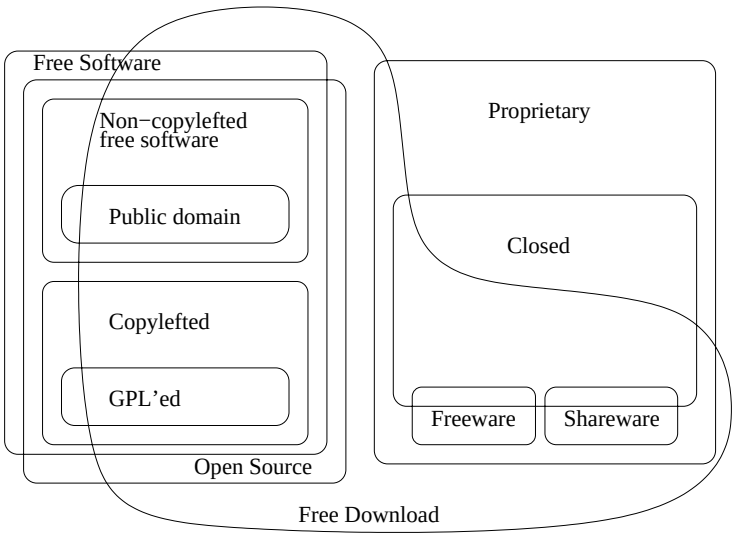

Figure 1: Categories of software

The classification of software in the manner above can be seen diagrammatically as Figure 1, which was adapted from the software categories based on the Free Software Foundation view [19]. Table 1 below summarises the main comparisons between the characteristics of those software categories.

There are subtle differences between open source and free software, in particular around licensing issues. For example, open source software may use proprietary library (e.g. the KDE project [20] was using a proprietary library called Qt until September 2000), which is unacceptable in free software. Further investigation surrounding these differences could provide better understanding, as highlighted in section 5.

Table 1: Comparisons of different kinds of software

	Open Source (Free) Software		**Proprietary Software**		
	Non-copylefted	*Copylefted*	*Closed*	*Shareware*	*Freeware*
Availability of source code	Y	Y	N	N	N
Permission to					
• redistribute	Y	Y	N	Y	Y
• modify	Y	Y	-	-	-
• add restriction	Y	N	N	N	N
Modified version always free	N	Y	-	-	-
Free Download	Y	Y	N	Y	Y
Time Limit in usage	N	N	N	Y	N
Possibility of making money	Y	Y	Y	Y	N

3 Characteristics of Open Source

By exposing the characteristics that open source projects usually have, we hope to be able to develop a clearer picture on what it really means for a particular project or software development to be an open source project[1] or not. The idea is to have a "tick-list" of open source characteristics, against which the characteristics of the project in question can be compared. Additionally, these characteristics highlight the fact that just stating that a project is open source does not necessarily provide a precise definition.

3.1 Disciplines to consider

In the spirit of DIRC[2], a research project that we are working on, it is important to highlight that software development is a very complex process that draws upon knowledge/expertise from many scientific disciplines. Therefore, to understand it better, it is necessary to emphasise its interdisciplinary nature. It appears that open source software development is no exception, and in order to determine the relevant open source characteristics, there are several disciplines that we would like to consider:

- Computing Science
  Covering the technical aspects that need to be considered to engage in an open source project.
- Management Issues
  Dealing with managerial issues and how they relate to open source projects.
- Social Sciences

[1] The term 'project' is used loosely in this paper, as it is doubtful whether OSS projects fulfil the more generic management definition of a unique/novel activity with explicit/finite timescales. Should the use of this term create conflicts of definition, for readers, they can interpret the term 'project' as 'undertakings' or 'initiatives'.

[2] DIRC is a UK EPSRC project based on a Dependable Interdisciplinary Research Collaboration (DIRC) on computer-based systems (see http://www.dirc.org.uk/).

Addressing areas related to the communities involved in open source projects and their behaviour.

- Psychology
  Accounting for the characteristics of the individuals involved in open source projects.
- Organisational Aspects
  Dealing with aspects such as organisational structures.
- Economics
  Looking into economic models that underlie open source projects and/or corporations with respect to their involvement in open source projects.
- Law
  Focusing on legal issues.

Clearly, the OSI definition for the term open source does address legal issues extensively, and encompasses some economic aspects. On the other hand, it hardly touches on computing science areas; it also completely ignores the areas of management, psychology, social sciences and organizational aspects. Furthermore, there is no guarantee that a given project, by simply adhering to the OSI definition of the term open source, benefits from the positive effects that are usually related to the term open source (e.g. being reviewed by many people). The open source software characteristics proposed by Wang and Wang [11] address some technical aspects, and in less depth, legal and managerial aspects.

In our attempt to understand open source, we determined a set of characteristics that occur under that umbrella term, while considering the various disciplines mentioned above. Some characteristics are common to all efforts we were able to investigate, whereas others vary between projects. The set of characteristics we deem relevant for discussing open source are described below, section 3.2 covering those that are common throughout open source projects and section 3.3 addressing those that vary between projects.

3.2 Common characteristics

Open source projects have many common characteristics. All items listed under the OSI definition of open source, OSD (see section 2.2), are the basic requirements for projects to qualify as open source. Moreover, *active* open source projects rely upon several other characteristics. We have identified six characteristics that are present in successful open source projects, these are addressed below.

Community

All active open source projects have a well-defined community with common interests that are either involved in continuously evolving its related products and/or in using its results. Anecdotally, the community, in its vast majority, is composed by men. Communications tend to be constructive, at times becoming confrontational.

Motivation

The biggest question surrounding the open source phenomena is *why do people do it?* What is the explanation behind having people providing contributions for free? The answer to these questions is not as straightforward as one might have thought. There are *different types of contributors*, individuals and corporations. Individuals usually

contribute for personal satisfaction; some have really strong philosophical beliefs others do not care as much about such issues. Corporations usually get involved with the aim to gain market share, undermine their competitors, or simply rely on products generated by open source without having to build a fully equivalent product from scratch.

Peer recognition also plays a role on motivating contributions. By having their contributions recognized as appropriate and of good quality by the community involved, both individuals and corporations have their status raised within the given project. Consequently, their opinions are considered more carefully with respect to project related decisions and their reputation may even improve outside the project boundaries.

Developers' profile

The set of people that contribute code to specific open source projects is always composed of those that are also users of the code produced. This means that open source developers are a subset of the open source user community, i.e. all open source developers are users, but not all users are developers (Figure 3).

This characteristic explains the fact that there are normally no precise specifications or requirements documents clarifying what is to be achieved in the project. It also highlights that it is quite unrealistic to expect the open source community to start developing arbitrary kinds of software. Software developers are usually not expert users of medical systems, nuclear plant control systems, or air traffic control systems.

Process of accepting submissions

An open source project evolves by receiving submissions from various sources to address various aspects of the project. The most common submissions are those of bug reports and source code, others include documentation and test cases. Furthermore, open source projects often post the areas in which they are interested in receiving submissions. As a consequence, multiple concurrent submissions may be received addressing the exact same area. Therefore, open source projects have in place processes for accepting various types of submissions, also making it clear on how to handle multiple concurrent submissions.

The process of accepting submissions is composed of two main parts: the *decision making process* and the *process of disseminating information on submissions*. How these two parts get implemented varies from one open source project to another (see section 3.3).

Development improvement cycles

Product improvement in the open source software development process can manifest in both breakthrough and continuous improvement modes. Breakthrough improvement involves dramatic and relatively impromptu changes [21]. Evidence of this form of product improvement in open source development was provided by Raymond [5] in the development of Fetchmail. He notes that:

> *"The real turning point in the project was when Harry Hochheiser sent me his scratch code for forwarding mail to the client machine's SMTP port…this SMTP-forwarding concept was the biggest single payoff I got…The Cruftiest*

parts of the driver vanished. Configuration got radically simpler…the only way to lose mail vanished…and performance improved." (p. 47-50)

Continuous improvement involves an increased frequency of change but in smaller and more incrementally consolidating stages [21]. This philosophy of product development recognises that small improvements build up to larger improvement overtime, but with the added advantage of being far easier to implement. Incremental product improvement through bug finding and fixing is a development hallmark of the open source paradigm and is embodied in Eric Raymond's original characterisation "release early, release often" [5] The idea is to get quick feedback, which can then be incorporated back into the product.

More recently such anecdotal claims have been further reinforced by the research findings of Aoki et al. with the open source *Jun* project [22]. They tracked the evolution of the software over 360 versions and identified both incremental improvements within single version updates followed by significant functionality increases requiring major modification to the existing architecture. Both of these forms of product improvement are generically shown in Figure 2 below.

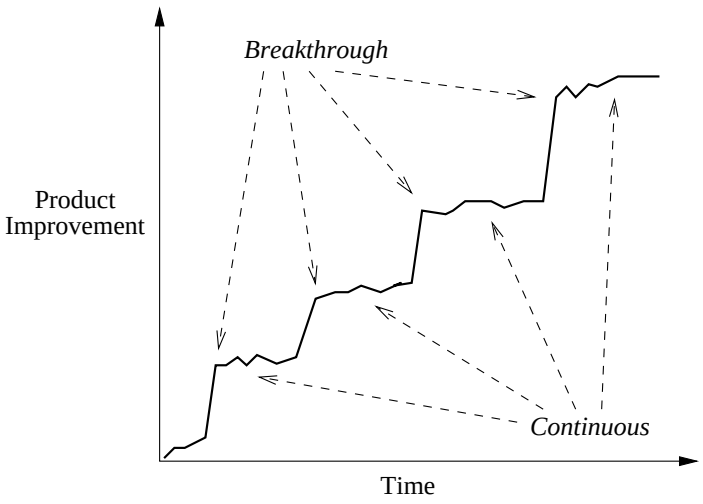

Figure 2: Open source product improvement over time.

Modularity

The benefits of modular design are well established in all engineering disciplines, as it supports increased understanding during design and concurrent allocation of work during implementation [23]. However, due to the globally distributed nature of open source development, well-defined interfaces and modularised source-code are a prerequisite for effective remote collaboration [24].

3.3 Variable characteristics

The areas in which open source projects vary are much more numerous than those that they have in common. Below is a discussion of some of those.

Choice of work area

As previously mentioned, open source projects often request contributions to the areas in which they are interested in receiving submissions. Some open source projects will

process both solicited and spontaneous contributions, whereas other open source projects may be prone to ignoring spontaneous contributions.

Balance of centralisation and decentralisation

The communities within various open source projects are organised differently. Some have a very strict hierarchy differentiating among various levels of developers (see Figure 3), whereas others have a much looser structure. The strict hierarchies bring with them a more centralised power structure, for example, the *core* developers have more power than ordinary (co-) developers in making executive decisions. In some open source projects (e.g. Apache), it is even possible to have more than two levels of developers. But not all open source projects have multi-level developer groups. Looser organisational structures have all their developers on the same level, which implies decentralisation of decisions, at times being based on full consensus for approving decisions.

Meritocratic culture

The basic model underlying open source projects is that knowledge shown by means of contributions increases the perception of merit, which in turn leads to power. Exactly how this transition takes place varies from project to project in terms of timing and the obstacles that must be overcome, and depends on the actual organisational structure of the project. For example, Figure 3 shows the possible transition from passive to active users when they start contributing to the project. If they could then show their ability (or they could gain respect from the community), they might be invited into the developer group, where they would have greater rights over the code (e.g. to incorporate their own modifications into the code base). In some projects, there is also a possibility of promotion from the co-developer to the core developer group. The transitions can also go the other way, e.g. a core developer might wish to resign and become a co-developer instead (or even leave the project completely) due to other commitments or personality clash.

Business model

Depending on the domain that an open source project addresses, different business models may motivate the involvement of commercial corporations, researchers, individual developers and end-users. The business models we have identified so far are: own use, packaging and selling, and platform/foundation for commercial or research software development.

Decision making process

The decision making process relies on four dimensions that vary from open source project to project. These are the *quality goals*, the *acceptance criteria* enacted, the *cognitive abilities* of the decision group, and the *social structure* within the project. Quality goals vary widely from one open source project to another; this can be observed even in the same application area (e.g. one focusing on performance and another on portability). The acceptance criteria used also vary among open source projects. It can be the best solution out of the first *n* submissions, some form of aggregation of multiple submissions (even by requesting that someone changes their solution to add some other aspect seen elsewhere), some memory of previous submissions by the same person, the first submission received, etc. Additionally, the

ability to recognise better solutions is highly dependent on the cognitive abilities of the decision group. This implies that the decision making process on accepting submissions varies among projects and potentially within projects as well, unless the same people are involved in all decisions.

The social structure inherent to an open source project may be a defined hierarchy where different groups of people get to evaluate different submissions (e.g. by focus area) and/or some people exercise greater power, or a monolithic group composed of all developers. The social structure impacts directly on the decision making process. If the group is monolithic then the acceptance of submissions may be achieved by consensus or majority voting. If there is some other form of social structure, the same consensus or majority voting may apply, at times with the votes of some of the members counting more than others.

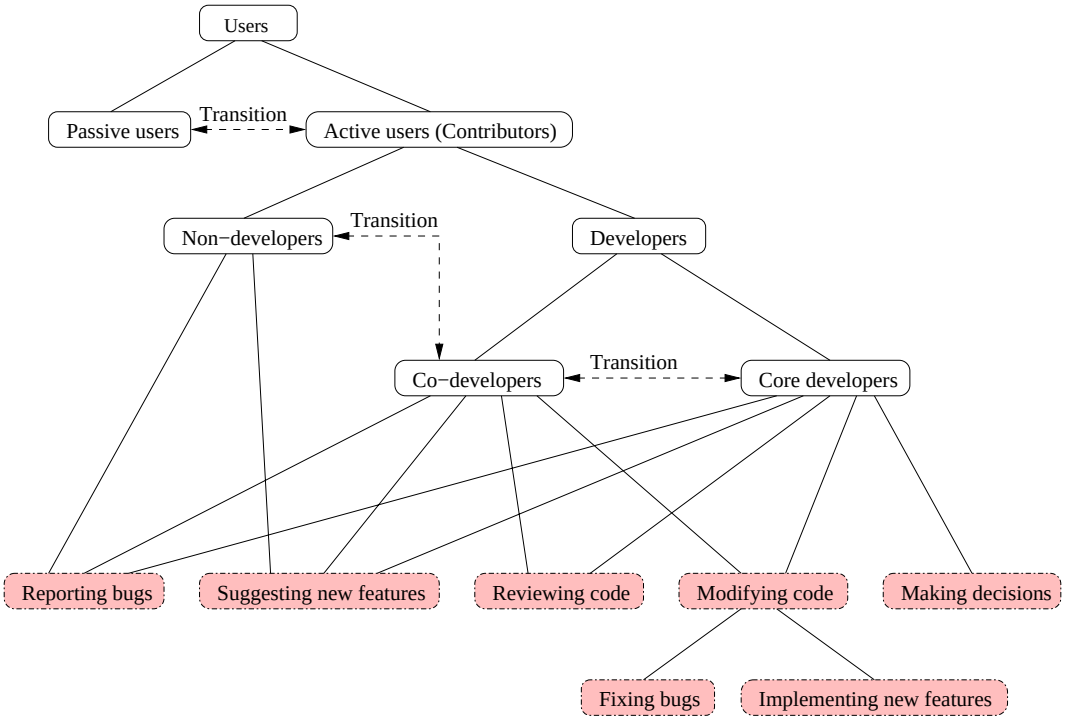

Figure 3: The classification of open source users and developers

Submission information dissemination process

The information on submissions and their acceptance may be passively disseminated by the means of newsgroups or comments in the code itself, it may be actively disseminated by using emails and mailing lists, or there may be some dedicated web space for statistical information.

Project starting points

Open source software projects may start from scratch or from existing closed source software systems, either commercial or research. From the various projects that we studied we could only find examples of projects that transitioned the full package from closed to open source at once. Nevertheless, one can envision some closed source software making a gradual transition to open source, one part (e.g. a subsystem) at a time.

Visibility of software architecture

The software architecture of a computing system depicts its structure(s) and comprises its software components, the externally visible properties of those components, and the relationships among them [25]. The architecture of an open source software system may be itself open or closed. The "closedness" may occur intentionally or accidentally. Having an intentionally closed software architecture means that the core group will consciously not reveal the structure to the general public. An unintentionally closed software architecture suggests that the structure exists in some people's minds only.

Documentation and testing

Documentation and testing are important aspects of the software development process. Good documentation allows people to use – and more specifically in open source projects, to understand and modify – the software. Thorough testing enables the users (and the developers) to have confidence that the software they are using (or developing) is going to function as expected.

These two areas are often overlooked or vary widely in the open source development process. Open source contributors tend to be more interested in coding than documenting or testing. This is probably due to the nature of open source that tries to replace the formal testing process with "many eyeballs" effect in eliminating the bugs. Also, adding comments in the source code is often perceived as sufficient for documentation. There has been some effort in addressing the problem of lack of documentation (e.g. the *Linux Documentation Project* [26] and *Mozilla Developer Documentation* web page [27]), but this is still a rarity for smaller open source projects. We have yet to find some sort of testing strategies for open source projects. They might exist, but implicitly and not open to the outside the project.

Licensing

The basic freedoms of open source software and how they differ from other software distributions were discussed in section 2.1 and 2.4 earlier. Here we consider the main varying features of OSD and FSF qualifying licenses[3]. Whether the software is viral or can become closed (proprietary) reflects the two main varying features of free and open source software.

Table 2 illustrates this with some of the more popular public licenses conforming to the OSD/FSF definitions. Viral licenses ensure that if any of the software code is used in other software developments then this will cause all of the software to come under the terms of that original license. The other varying feature

Table 2: Varying characteristics of open source licenses

Licenses	Is it viral?	Can it be closed?
GPL	Yes	No
LGPL	No	No
BSD	No	Yes
Q Public	No	No
IBM	No	Yes
Netscape (i.e. Mozilla)	No	Yes

[3] The term 'qualifying' refers to the four fundamental freedoms that both the OSD and FSF agree on.

concerns whether the license allows any of the original source code to be distributed in binary form only in future derived software products.

Operational support

In order to facilitate concurrent software development and fast controlled evolution, all open source projects implement some form of configuration management. This is enacted by using CVS, other tools, or even an ad-hoc solution using some web-based support.

The communication within communities related to specific open source projects is done almost exclusively by electronic means, which are also used to organise their work. The electronic means most commonly used are dedicated mailing lists, newsgroups, and web site. The exact structure and usage of web sites, mailing lists and newsgroups vary among open source projects.

Size

Size is not a distinctive measure in open source projects. Both involved-community and code base sizes vary widely from project to project.

4 Conclusion

The term open source is being used within the computing science community at large in a vague manner, consequently creating confusion and misunderstandings. In our efforts to understand open source we have done an extensive literature review, explored several web sites related to the topic, and interviewed some individuals and

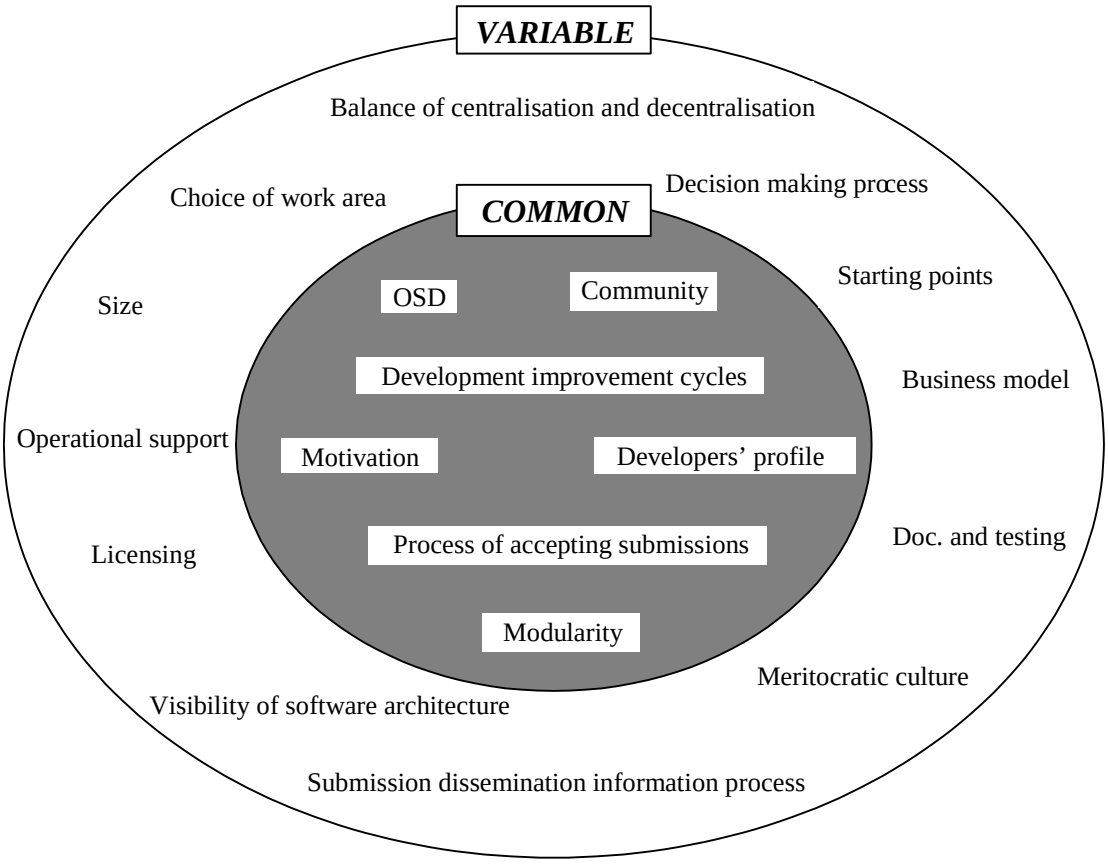

Figure 4: Open source characteristics – common and variable

corporations involved with open source. Our work was performed bearing multiple disciplines in mind.

We have determined many project characteristics that are relevant for open source. Some of these characteristics are common to all efforts, whereas others vary among open source projects (Figure 4).

How the various characteristics relate to the disciplines discussed in section 3.1 is highlighted in Table 3.

The set of open source characteristics we found can be used as a tick-list both for analysing and for setting up open source projects. We understand that there is no way that an absolute tick-list can ever be generated due to the variations that exist from one open source project to another, so additional variable characteristics may exist. Our proposed tick-list provides a starting point for understanding open source and its many meanings.

Table 3: Open source characteristics and disciplines considered

	Computing Science	Management Issues	Social Sciences	Psychology	Organizational Aspects	Economics	Law
OSD	√					√	√
Community			√	√			
Motivation		√	√	√		√	
Developers' profile	√		√		√		
Process of accepting submissions	√	√			√		
Development improvement cycles	√	√	√				
Modularity	√	√		√			
Choice of work area	√	√		√			
Balance of centralisation and decentralisation		√			√		
Meritocratic culture			√		√		
Business model						√	
Decision making process	√	√	√	√			
Submission information dissemination process		√	√				
Project starting points	√	√				√	
Visibility of software architecture	√	√	√	√	√		
Documentation and testing	√	√	√				
Licensing						√	√
Operational support	√	√	√		√		
Size	√		√		√	√	

Our work has led us to understand that it would be unreasonable to try to discuss open source software in general. There are as many differences among open source software projects as among non-open source software projects. Furthermore, many of the characteristics present in open source software projects are not restricted to open source software environments, they may also be found in some proprietary environments. Simply using the term open source is not enough, just as using the term proprietary software does not suffice.

Consequently, discussions comparing software project processes and approaches ought to occur at a lower level of granularity, at the individual characteristics level, in order to be fruitful. Whether projects are more or less successful, or exhibit a lower or higher expected quality, depends on the characteristics of the development and maintenance environment that they are in.

5 Future Work

There are many issues still left to be investigated with respect to understanding and exploiting the open source approach. Future work should further clarify the exact differences between open source and free software, as well as generate a table relating various existing open source and free software projects to the characteristics we set forth, while describing how each of these projects implement the variable parts.

There are also *dependability* issues that need to be addressed. We shall be looking into statistical information, such as bug density, fixing time, hacking incidents, etc., regarding open source software, free software, and proprietary software. This shall be done by grouping software packages according to their individual characteristics, rather than by grouping them under the labels that we have just used above (open source, free and proprietary software), with the aim of determining which openness characteristics foster more dependable systems or not.

6 Acknowledgements

This paper has been funded by the UK EPSRC project on Dependable Interdisciplinary Research Collaboration (DIRC – http://www.dirc.org.uk/). We would like to thank the volunteers – in particular, Julian Coleman, Stuart Wheater, and Mike Ellison – that spent their time while sharing their experiences with us. We would also like to thank our colleagues from the DIRC project involved in the Open Source activity for various fruitful discussions contributing towards this paper.

7 References

[1] "The Open Source Initiative: Open Source Definition", http://www.opensource.org/docs/definition.html.

[2] "The Linux Home Page at Linux Online", http://www.linux.org/.

[3] "The Apache Software Foundation", http://www.apache.org.

[4] "mozilla.org", http://www.mozilla.org/.

[5] E. S. Raymond, *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*, O'Reilly & Associates, 1999.

[6] G. Moody, *Rebel Code: Linux and the Open Source Revolution*, The Penguin Press, 2001.

[7] C. Dibona, M. Stone, and S. Ockman, *Open Sources: Voices from the Open Source Revolution*, O'Reilly & Associates, 1999.

[8] A. Mockus, R. T. Fielding, and J. Herbsleb, “A Case Study of Open Source Software Development: The Apache Server,” Proceedings of ICSE 2000, pp. 263-272, 2000.
[9] M. W. Godfrey and Q. Tu, “Evolution in Open Source Software: A Case Study,” Proceedings of International Conference on Software Maintenance (ICSM'00), 2000.
[10] B. J. Dempsey, D. Weiss, P. Jones, and J. Greenberg, “A Quantitative Profile of a Community of Open Source Linux Developers”, SILS TR-1999-05, 1999.
[11] H. Wang and C. Wang, “Open Source Software Adoption: A Status Report,” *IEEE Software*, March/April, pp. 90-95, 2001.
[12] J. Feller and B. Fitzgerald, “A framework analysis of the open source software development paradigm,” Proceedings of 21st International Conference on Information Systems, pp. 58-69, 2000.
[13] “SourceForge”, http://sourceforge.net/.
[14] “Geocrawler”, http://www.geocrawler.org/.
[15] “The Free Software Definition - GNU Project - Free Software Foundation (FSF)”, http://www.fsf.org/philosophy/free-sw.html.
[16] “The Open Source Initiatiative: History of the OSI”, http://opensource.org/docs/history.html.
[17] “Why Free Software is better than Open Source”, http://gnu.metagensoft.com/philosophy/free-software-for-freedom.html.
[18] E. S. Raymond, “Homesteading the Noosphere”, http://tuxedo.org/~esr/writings/homesteading/homesteading/.
[19] “Categories of Free and Non-Free Software”, http://www.gnu.org/philosophy/categories.html.
[20] “K Desktop Environment Home”, http://www.kde.org/.
[21] N. Slack, S. Chambers, C. Harland, A. Harrison, and R. Johnston, *Operations Management*, 2nd ed, Financial Times Pitman Publishing Series, 1998.
[22] A. Aoki, K. Hayashi, K. Kishida, K. Nakakoji, Y. Nishinaka, B. Reeves, A.Takashima, and Y. Yamamoto, “A Case Study of the Evolution of Jun: an Object-Oriented Open-Source 3D Multimedia Library,” Proceedings of 23rd ICSE Conference, Toronto, Canada, pp. 524-532, 2001.
[23] R. N. Britcher, *The Limits of Software: People, Projects, and Perspectives*, Addison Wesley, 1999.
[24] T. Bollinger, R. Nelson, K. M. Self, and S. J. Turnbull, “Open-Source Methods: Peering Through the Clutter,” *IEEE Software*, July/August, pp. 8-11, 1999.
[25] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, Addison Wesley, 1998.
[26] “Linux Documentation Project”, http://www.linuxdoc.org.
[27] “Mozilla Developer Documentation”, http://www.mozilla.org/docs/.